```
!pip install opencv-python scikit-image n2v csbdeep
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Collecting n2v
  Downloading n2v-0.3.3-py2.py3-none-any.whl.metadata (8.6 kB)
Collecting csbdeep
  Downloading csbdeep-0.8.1-py2.py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.15.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (3.4.2)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (11.2.1)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2025.3.30)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (0.4)
Collecting imagecodecs>=2020.2.18 (from n2v)
  Downloading imagecodecs-2025.3.30-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (20 kB)
Collecting csbdeep
  Downloading csbdeep-0.7.4-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting ruamel.yaml>=0.16.10 (from n2v)
  Downloading ruamel.yaml-0.18.10-py3-none-any.whl.metadata (23 kB)
Collecting bioimageio.core (from n2v)
  Downloading bioimageio_core-0.8.0-py3-none-any.whl.metadata (23 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from csbdeep) (3.10.0)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from csbdeep) (1.17.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from csbdeep) (4.67.1)
Collecting ruamel.yaml.clib>=0.2.7 (from ruamel.yaml>=0.16.10->n2v)
  Downloading ruamel.yaml.clib-0.2.12-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.7 kB)
Collecting bioimageio.spec==0.5.4.1 (from bioimageio.core->n2v)
  Downloading bioimageio.spec-0.5.4.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from bioimageio.core->n2v) (3.13.0)
Collecting loguru (from bioimageio.core->n2v)
  Downloading loguru-0.7.3-py3-none-any.whl.metadata (22 kB)
Collecting pydantic-settings<3,>=2.5 (from bioimageio.core->n2v)
  Downloading pydantic_settings-2.9.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: pydantic<3,>=2.7.0 in /usr/local/lib/python3.11/dist-packages (from bioimageio.core->n2v) (2.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from bioimageio.core->n2v) (2.32.3)
Collecting ruyaml (from bioimageio.core->n2v)
  Downloading ruyaml-0.91.0-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from bioimageio.core->n2v) (4.13.2)
Collecting xarray<2025.3.0,>=2023.01 (from bioimageio.core->n2v)
  Downloading xarray-2025.1.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: annotated-types<1,>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.
Collecting email-validator (from bioimageio.spec==0.5.4.1->bioimageio.core->n2v)
  Downloading email_validator-2.2.0-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: markdown in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.1->bioimageio.cor
Requirement already satisfied: pooch<2,>=1.5 in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.1->bioimagei
Collecting pydantic<3,>=2.7.0 (from bioimageio.core->n2v)
  Downloading pydantic-2.9.2-py3-none-any.whl.metadata (149 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.4/149.4 kB 5.3 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.1->bioimag
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.1->bioimageio.core->n
Requirement already satisfied: zipp in /usr/local/lib/python3.11/dist-packages (from bioimageio.spec==0.5.4.1->bioimageio.core->n
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->csbdeep) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->csbdeep) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->csbdeep) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->csbdeep) (1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->csbdeep) (3.2.3)
```

```
!pip install PyWavelets # Install the missing PyWavelets package
```

```
Collecting PyWavelets
  Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.0 kB)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from PyWavelets) (2.0.2)
Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 37.6 MB/s eta 0:00:00
Installing collected packages: PyWavelets
Successfully installed PyWavelets-1.8.0
```

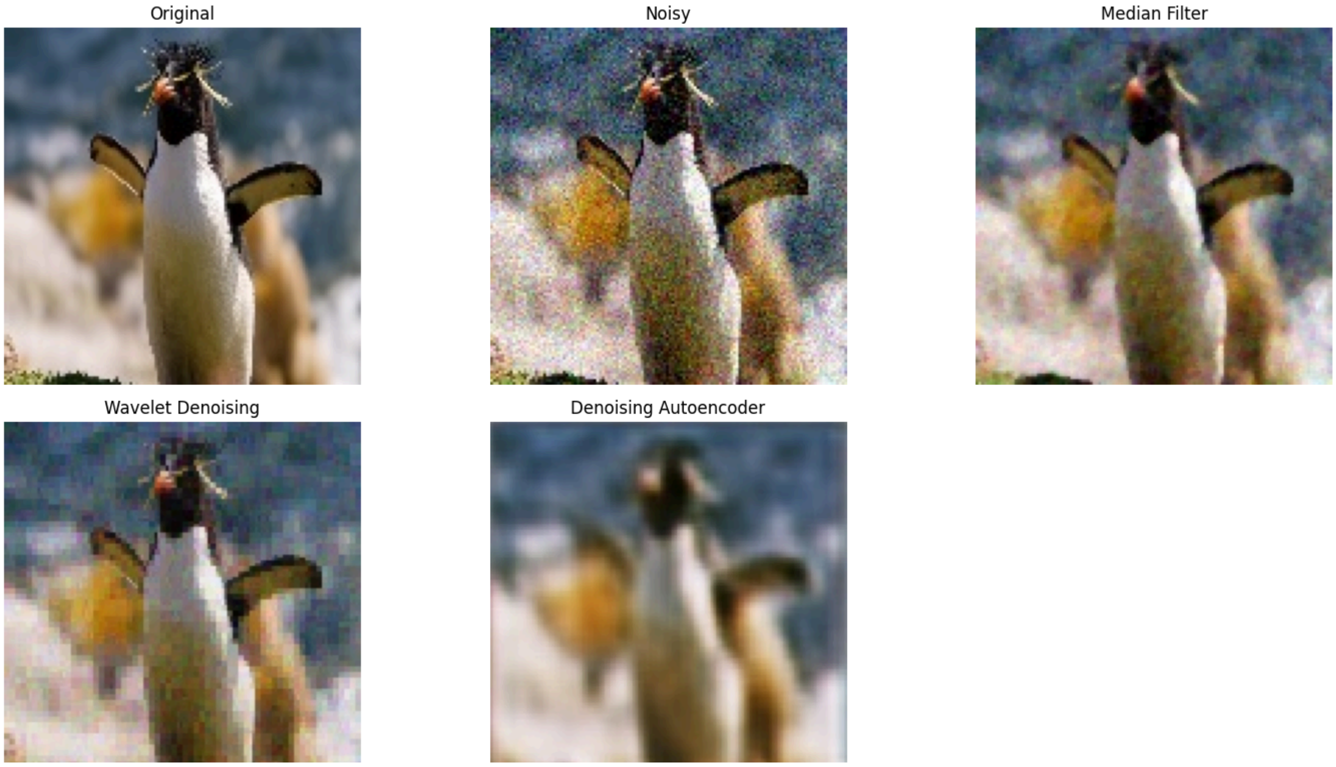
```
# Install required packages
!pip install scikit-image opencv-python-headless tensorflow matplotlib --quiet

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import img_as_float
from skimage.restoration import denoise_wavelet
from skimage.metrics import peak_signal_noise_ratio, structural_similarity, mean_squared_error
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import img_to_array, array_to_img

# Create folders
os.makedirs("original_images", exist_ok=True)
os.makedirs("denoised_results", exist_ok=True)

# Utility: Add synthetic noise to image
def add_noise(img, noise_type="gaussian"):
    row, col, ch = img.shape
    if noise_type == "gaussian":
        mean = 0
        sigma = 25
        gauss = np.random.normal(mean, sigma, (row, col, ch)).reshape(row, col, ch)
        noisy = img + gauss
        return np.clip(noisy, 0, 255).astype(np.uint8)
    return img

# Utility: Denoising Autoencoder
def build_denoising_autoencoder(input_shape):
    input_img = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)

    x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')
    return autoencoder

# Main processing function
def denoise_and_compare(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (128, 128))  # Resize for DAE
    noisy_img = add_noise(img_resized).astype(np.uint8)

    # Convert to float
    img_float = img_as_float(noisy_img)

    # 1. Median Filter
    median = cv2.medianBlur(noisy_img, 3)

    # 2. Wavelet Denoising
    wavelet = denoise_wavelet(img_float, channel_axis=-1, rescale_sigma=True)
    wavelet_uint8 = (np.clip(wavelet, 0, 1) * 255).astype(np.uint8)

    # 3. Denoising Autoencoder
    x_train = np.array([noisy_img]) / 255.0
    y_train = np.array([img_resized]) / 255.0
    dae = build_denoising_autoencoder(input_shape=(128, 128, 3))
    dae.fit(x_train, y_train, epochs=100, verbose=0)
    dae_output = dae.predict(x_train)[0]
    dae_output_uint8 = (dae_output * 255).astype(np.uint8)

    # Save outputs
    cv2.imwrite("denoised_results/original.jpg", cv2.cvtColor(img_resized, cv2.COLOR_RGB2BGR))
    cv2.imwrite("denoised_results/noisy.jpg", cv2.cvtColor(noisy_img, cv2.COLOR_RGB2BGR))
    cv2.imwrite("denoised_results/median.jpg", cv2.cvtColor(median, cv2.COLOR_RGB2BGR))
    cv2.imwrite("denoised_results/wavelet.jpg", cv2.cvtColor(wavelet_uint8, cv2.COLOR_RGB2BGR))
    cv2.imwrite("denoised_results/dae.jpg", cv2.cvtColor(dae_output_uint8, cv2.COLOR_RGB2BGR))

    # Metric calculations
    def metrics(original, filtered):
        return {
            "PSNR": peak_signal_noise_ratio(original, filtered),
            "SSIM": structural_similarity(original, filtered, channel_axis=2),
            "MSE": mean_squared_error(original, filtered)
        }

    print("=== Image Denoising Metrics ===")
    print("Median Filter:", metrics(img_resized, median))
    print("Wavelet Filter:", metrics(img_resized, wavelet_uint8))
    print("Denoising Autoencoder:", metrics(img_resized, dae_output_uint8))

# Upload and process one image
from google.colab import files
```

```
uploaded = files.upload()
import shutil
for filename in uploaded:
    shutil.move(filename, f"original_images/{filename}")

# Denoise the uploaded image
img_path = os.listdir("original_images")[0]
denoise_and_compare(f"original_images/{img_path}")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving i3.jpg to i3 (1).jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
=== Image Denoising Metrics ===
Median Filter: {'PSNR': np.float64(25.133760693201673), 'SSIM': np.float64(0.717523015893685), 'MSE': np.float64(199.39042154947916
Wavelet Filter: {'PSNR': np.float64(25.5084925478363), 'SSIM': np.float64(0.7431811421219038), 'MSE': np.float64(182.9073282877604)
```

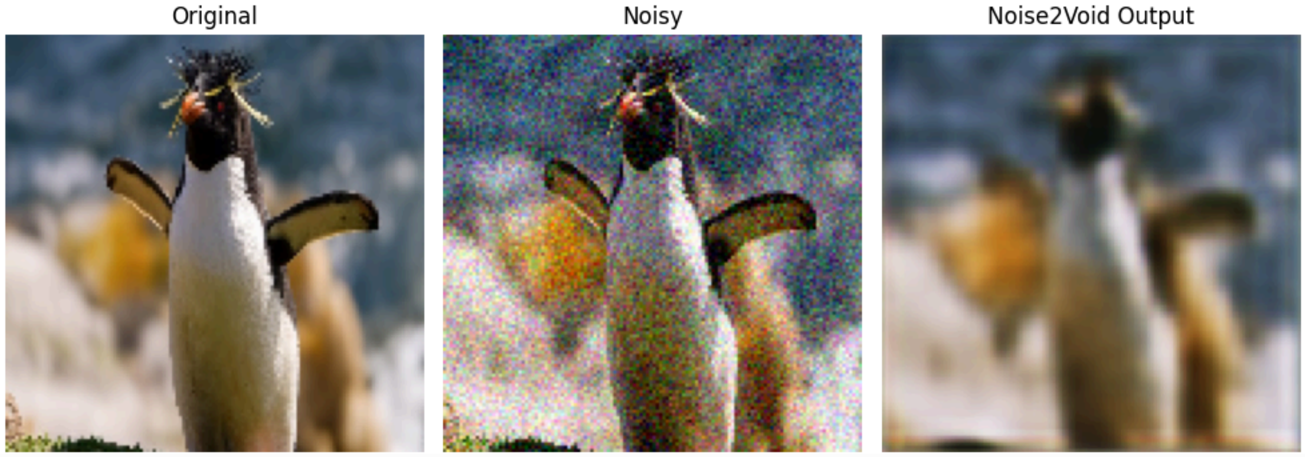
```
# Visualization of results
def show_results():
    titles = ['Original', 'Noisy', 'Median Filter', 'Wavelet Denoising', 'Denoising Autoencoder']
    files = ['original.jpg', 'noisy.jpg', 'median.jpg', 'wavelet.jpg', 'dae.jpg']
    plt.figure(figsize=(15, 8))

    for i, (title, file) in enumerate(zip(titles, files)):
        img = cv2.imread(f'denoised_results/{file}')
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(2, 3, i+1)
        plt.imshow(img)
        plt.title(title)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Call this function after denoising
show_results()
```



```
# Load image
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_path = "/content/i3.jpg"  # update with your actual filename
img = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_resized = cv2.resize(img_rgb, (128, 128))  # Resize for autoencoder
```

```
# Add synthetic Gaussian noise
noisy_img = add_noise(img_resized).astype(np.uint8)

# Normalize and prepare input
input_img = np.expand_dims(noisy_img / 255.0, axis=0)

# Build and train autoencoder (simulating Noise2Void behavior)
autoencoder = build_denoising_autoencoder((128, 128, 3))
autoencoder.fit(input_img, input_img, epochs=100, verbose=0)  # Train on noisy image only

# Predict denoised output
denoised = autoencoder.predict(input_img)[0]
denoised_uint8 = (denoised * 255).astype(np.uint8)

# Save outputs
cv2.imwrite("denoised_results/noise2void_input.jpg", cv2.cvtColor(noisy_img, cv2.COLOR_RGB2BGR))
cv2.imwrite("denoised_results/noise2void_output.jpg", cv2.cvtColor(denoised_uint8, cv2.COLOR_RGB2BGR))

# Show comparison
plt.figure(figsize=(10, 4))
plt.subplot(1, 3, 1)
plt.title("Original")
plt.imshow(img_resized)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Noisy")
plt.imshow(noisy_img)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Noise2Void Output")
plt.imshow(denoised_uint8)
plt.axis('off')

plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
```



```
# Install OpenCV if not already installed
!pip install opencv-python-headless --quiet

import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 1: Upload video
uploaded = files.upload()
video_file = list(uploaded.keys())[0]

# Create output directories
os.makedirs("frames", exist_ok=True)
os.makedirs("processed_videos", exist_ok=True)
os.makedirs("collage_frames", exist_ok=True)

# Step 2: Extract Frames from Video
cap = cv2.VideoCapture(video_file)
frame_count = 0
frames = []
```

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_path = f"frames/frame_{frame_count:04d}.jpg"
    cv2.imwrite(frame_path, frame)
    frames.append(frame)
    frame_count += 1

cap.release()
print(f"Total frames extracted: {frame_count}")
```

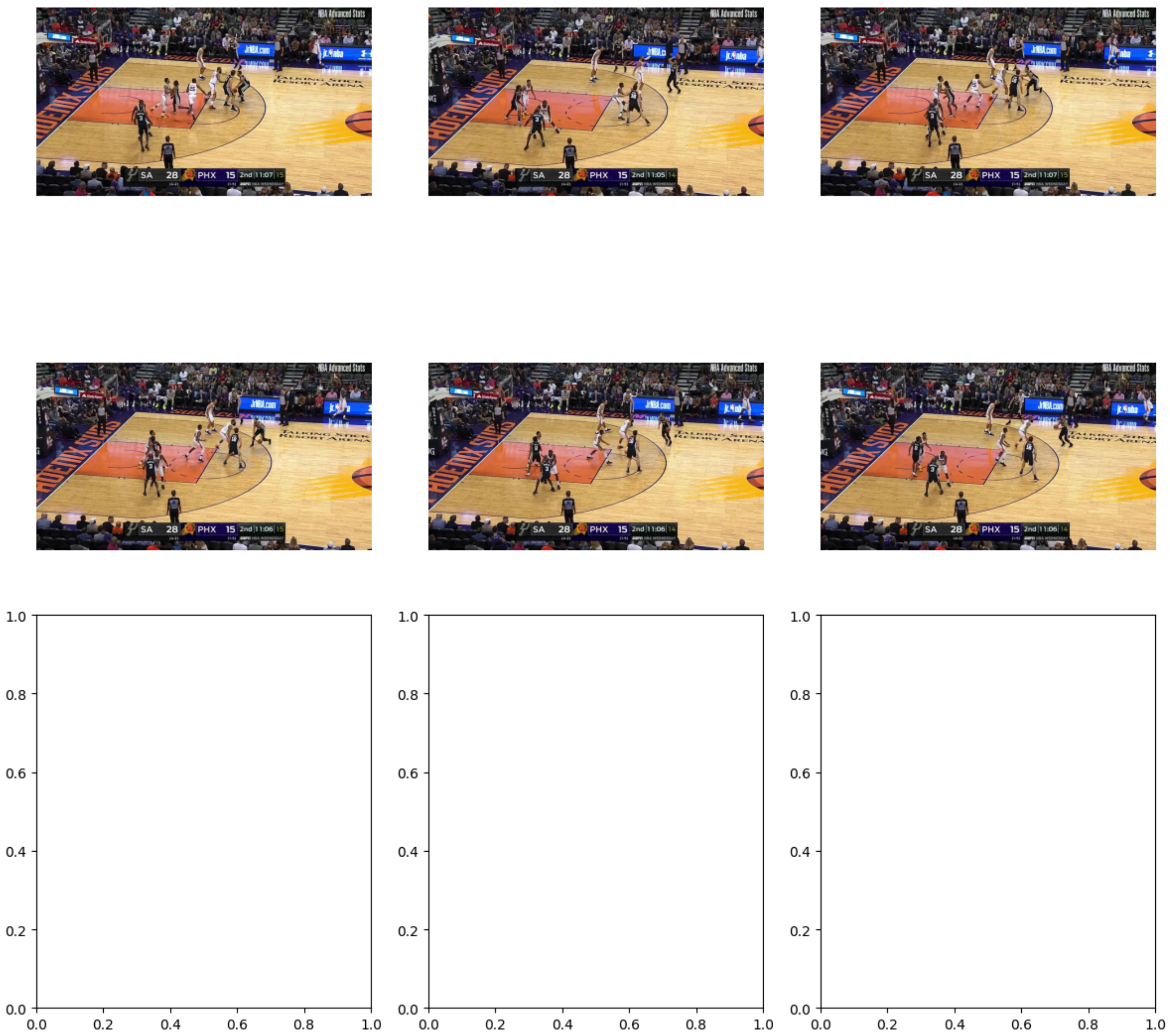
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 0021800110_154.mp4 to 0021800110_154 (1).mp4
```

```
# Define video writer settings
height, width, layers = frames[0].shape
fourcc = cv2.VideoWriter_fourcc(*'mp4v')

# Define output writers
out_thresh = cv2.VideoWriter("processed_videos/adaptive_threshold.mp4", fourcc, 20.0, (width, height), False)
out_blur = cv2.VideoWriter("processed_videos/gaussian_blur.mp4", fourcc, 20.0, (width, height), True)
out_edges = cv2.VideoWriter("processed_videos/canny_edges.mp4", fourcc, 20.0, (width, height), False)
out_not = cv2.VideoWriter("processed_videos/bitwise_not.mp4", fourcc, 20.0, (width, height), True)

for idx, frame in enumerate(frames):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Adaptive Thresholding
    thresh = cv2.adaptiveThreshold(gray, 255,
                                   cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)
    out_thresh.write(thresh)

    # Gaussian Blur
    blur = cv2.GaussianBlur(frame, (15, 15), 0)
    out_blur.write(blur)

    # Canny Edge Detection
    edges = cv2.Canny(gray, 100, 200)
    out_edges.write(edges)

    # Bitwise NOT
    bitwise_not = cv2.bitwise_not(frame)
    out_not.write(bitwise_not)

    # Save some frames for collage
    if idx % 20 == 0 and idx <= 100:
        cv2.imwrite(f"collage_frames/frame_{idx}.jpg", frame)

# Release video writers
out_thresh.release()
out_blur.release()
out_edges.release()
out_not.release()

print("Processed videos saved in 'processed_videos/' folder.")
```

```
Processed videos saved in 'processed_videos/' folder.
```

```
import glob

collage_images = sorted(glob.glob("collage_frames/*.jpg"))[:9]
collage_frames = [cv2.cvtColor(cv2.imread(img), cv2.COLOR_BGR2RGB) for img in collage_images]

fig, axes = plt.subplots(3, 3, figsize=(12, 12))
for ax, img in zip(axes.flatten(), collage_frames):
    ax.imshow(img)
    ax.axis('off')

plt.suptitle("Collage of Sample Frames", fontsize=16)
plt.tight_layout()
plt.show()
```



Collage of Sample Frames

```
import kagglehub
import os

# Step 1: Download the dataset
path = kagglehub.dataset_download("pevogam/ucf101")

# Step 2: Explore the downloaded files and directories
print("Base download path:", path)
print("\nContents of the downloaded path:")
print(os.listdir(path))

# Step 3: Search for the UCF101 video classes
for root, dirs, files in os.walk(path):
    print(f"\nFound directory: {root}")
    for d in dirs:
        print("  └──", d)
    break  # only print the first level
```

```
Base download path: /kaggle/input/ucf101

Contents of the downloaded path:
['UCF101', 'UCF101TrainTestSplits-RecognitionTask']

Found directory: /kaggle/input/ucf101
  └── UCF101
  └── UCF101TrainTestSplits-RecognitionTask
```

```
import os

root_dir = "/kaggle/input/ucf101/UCF101"
for root, dirs, files in os.walk(root_dir):
    print("Directory:", root)
    for d in dirs:
        print("  Subfolder:", d)
    for f in files[:3]:  # print only a few files for brevity
        print("  File:", f)
    print("-" * 40)
```

```
Directory: /kaggle/input/ucf101/UCF101
  Subfolder: UCF-101
----------------------------------------
Directory: /kaggle/input/ucf101/UCF101/UCF-101
  Subfolder: HorseRace
  Subfolder: StillRings
  Subfolder: ApplyLipstick
  Subfolder: HammerThrow
  Subfolder: VolleyballSpiking
  Subfolder: Biking
  Subfolder: PlayingCello
  Subfolder: BodyWeightSquats
  Subfolder: TaiChi
  Subfolder: Punch
  Subfolder: BreastStroke
  Subfolder: Billiards
  Subfolder: BoxingPunchingBag
  Subfolder: BasketballDunk
  Subfolder: PoleVault
  Subfolder: ThrowDiscus
  Subfolder: BaseballPitch
  Subfolder: Knitting
  Subfolder: SumoWrestling
  Subfolder: HorseRiding
  Subfolder: Mixing
  Subfolder: BrushingTeeth
  Subfolder: HighJump
  Subfolder: Skijet
  Subfolder: SkateBoarding
  Subfolder: MilitaryParade
  Subfolder: IceDancing
  Subfolder: CricketShot
  Subfolder: Fencing
  Subfolder: JugglingBalls
  Subfolder: Swing
  Subfolder: RockClimbingIndoor
  Subfolder: PlayingFlute
  Subfolder: SalsaSpin
  Subfolder: CricketBowling
  Subfolder: Typing
  Subfolder: ApplyEyeMakeup
  Subfolder: PlayingTabla
  Subfolder: BalanceBeam
  Subfolder: FloorGymnastics
  Subfolder: HeadMassage
  Subfolder: FrisbeeCatch
  Subfolder: Rowing
  Subfolder: Hammering
  Subfolder: CuttingInKitchen
  Subfolder: BenchPress
  Subfolder: PushUps
  Subfolder: Nunchucks
  Subfolder: Archery
  Subfolder: LongJump
  Subfolder: BlowingCandles
  Subfolder: WallPushups
  Subfolder: PlayingViolin
  Subfolder: PullUps
```

```
import os
import shutil
import random

# Define source and destination directories
SOURCE_DIR = '/kaggle/input/ucf101/UCF101/UCF-101'
DEST_DIR = '/kaggle/working/UCF101_subset'

# List of selected classes (can be updated as needed)
SELECTED_CLASSES = ['Basketball', 'Biking', 'PlayingGuitar', 'Typing', 'JumpRope']
VIDEOS_PER_CLASS = 10

# Create the destination directory if it doesn't exist
os.makedirs(DEST_DIR, exist_ok=True)

# Iterate over the selected classes and copy videos
```

```
for cls in SELECTED_CLASSES:
    class_path = os.path.join(SOURCE_DIR, cls)
    dest_class_path = os.path.join(DEST_DIR, cls)

    # Create class folder in destination
    os.makedirs(dest_class_path, exist_ok=True)

    # Select random 10 videos from the class
    selected = random.sample(os.listdir(class_path), VIDEOS_PER_CLASS)

    # Copy selected videos to the destination
    for video in selected:
        shutil.copy(os.path.join(class_path, video), dest_class_path)

print(f"Subset created at: {DEST_DIR}")
```

```
Subset created at: /kaggle/working/UCF101_subset
```

```
import cv2
import os
import numpy as np

# Define parameters
FRAME_RATE = 5  # Extract every 5th frame
RESIZE_DIM = (112, 112)  # Resize frames to 112x112
MAX_FRAMES = 16  # Number of frames per video

# Function to extract frames from video
def extract_frames(video_path, max_frames=MAX_FRAMES, frame_rate=FRAME_RATE, resize_dim=RESIZE_DIM):
    # Read the video
    cap = cv2.VideoCapture(video_path)

    frames = []
    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Extract every 'frame_rate'-th frame
        if frame_count % frame_rate == 0:
            frame_resized = cv2.resize(frame, resize_dim)
            frames.append(frame_resized)

        frame_count += 1

        # Stop once we have extracted enough frames
        if len(frames) == max_frames:
            break

    cap.release()

    # If fewer than MAX_FRAMES are extracted, pad the sequence with the last frame
    while len(frames) < max_frames:
        frames.append(frames[-1])

    return np.array(frames)

# Process each class and video
video_frames = {}
for cls in SELECTED_CLASSES:
    class_path = os.path.join(DEST_DIR, cls)
    video_frames[cls] = []

    for video in os.listdir(class_path):
        video_path = os.path.join(class_path, video)
        frames = extract_frames(video_path)
        video_frames[cls].append(frames)

print("Frame extraction completed.")
```

```
Frame extraction completed.
```

```
from sklearn.preprocessing import LabelEncoder

# Initialize label encoder
label_encoder = LabelEncoder()

# Fit the encoder on the selected classes
```

```
labels = label_encoder.fit_transform(SELECTED_CLASSES)

# Create a dictionary to map class names to labels
class_labels = dict(zip(SELECTED_CLASSES, labels))
print("Label encoding completed.")
```

```
Label encoding completed.
```

```
from sklearn.model_selection import train_test_split

# Prepare data and labels
data = []
labels = []

# Add frames and their corresponding labels
for cls in SELECTED_CLASSES:
    for frames in video_frames[cls]:
        data.append(frames)
        labels.append(class_labels[cls])

# Convert to numpy arrays
data = np.array(data)
labels = np.array(labels)

# Split data into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")
```

```
Training data shape: (40, 16, 112, 112, 3)
Testing data shape: (10, 16, 112, 112, 3)
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define 3D CNN Model
def create_3d_cnn_model(input_shape, num_classes):
    model = models.Sequential()

    # 3D convolution layers with padding='same'
    model.add(layers.Conv3D(32, kernel_size=(3, 3, 3), activation='relu', input_shape=input_shape, padding='same'))
    model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(layers.Conv3D(64, kernel_size=(3, 3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(layers.Conv3D(128, kernel_size=(3, 3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))

    # Flatten and fully connected layers
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Define input shape based on frame size and sequence length
input_shape = (MAX_FRAMES, RESIZE_DIM[0], RESIZE_DIM[1], 3)  # (16, 112, 112, 3) for 16 frames of 112x112 images with 3 color channels
num_classes = len(SELECTED_CLASSES)

# Create the model
model = create_3d_cnn_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Model summary
model.summary()
```

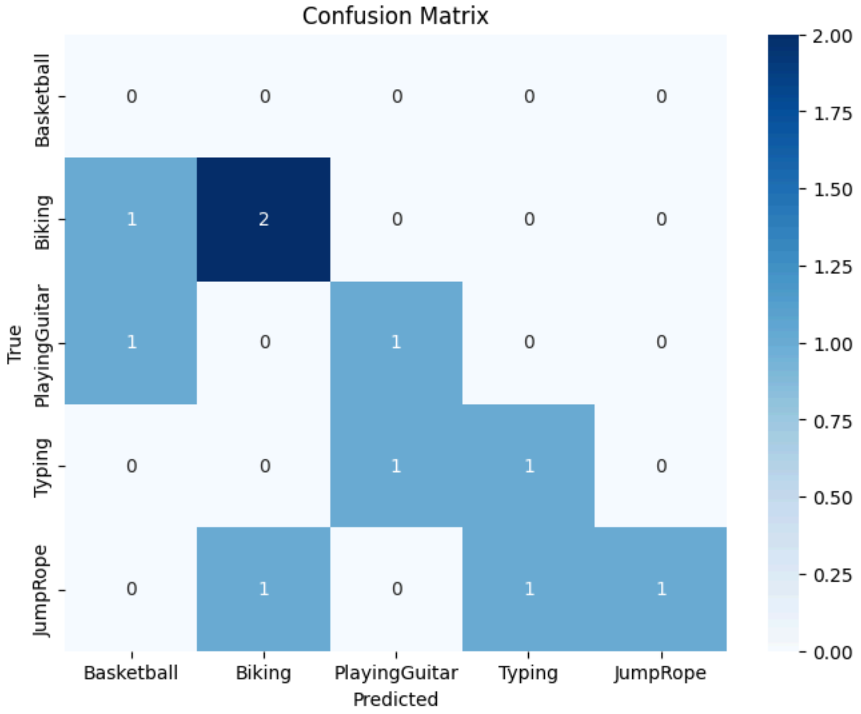
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv3d_3 (Conv3D)               │ (None, 16, 112, 112,   │         2,624 │
│                                 │ 32)                    │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling3d_3 (MaxPooling3D)  │ (None, 8, 56, 56, 32)  │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv3d_4 (Conv3D)               │ (None, 8, 56, 56, 64)  │        55,360 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling3d_4 (MaxPooling3D)  │ (None, 4, 28, 28, 64)  │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv3d_5 (Conv3D)               │ (None, 4, 28, 28, 128) │       221,312 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling3d_5 (MaxPooling3D)  │ (None, 2, 14, 14, 128) │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 50176)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 128)            │     6,422,656 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 5)              │           645 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 6,702,597 (25.57 MB)
 Trainable params: 6,702,597 (25.57 MB)
```

```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=4, validation_data=(X_test, y_test))

# Save the model
model.save("/kaggle/working/ucf101_3dcnn_model.h5")

print("Training completed and model saved.")
```

```
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 53s 5s/step - accuracy: 0.1854 - loss: 398.4858 - val_accuracy: 0.4000 - val_loss: 1.5153
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 81s 5s/step - accuracy: 0.2454 - loss: 1.5007 - val_accuracy: 0.2000 - val_loss: 2.3985
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 52s 5s/step - accuracy: 0.5691 - loss: 1.3393 - val_accuracy: 0.1000 - val_loss: 1.5773
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 52s 5s/step - accuracy: 0.6785 - loss: 1.1525 - val_accuracy: 0.3000 - val_loss: 2.2342
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 80s 5s/step - accuracy: 0.6778 - loss: 1.4211 - val_accuracy: 0.3000 - val_loss: 2.1582
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 83s 5s/step - accuracy: 0.5804 - loss: 0.9205 - val_accuracy: 0.1000 - val_loss: 1.7314
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 83s 5s/step - accuracy: 0.8245 - loss: 0.6209 - val_accuracy: 0.3000 - val_loss: 1.7927
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 82s 5s/step - accuracy: 0.7939 - loss: 0.5056 - val_accuracy: 0.6000 - val_loss: 2.0012
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 79s 5s/step - accuracy: 0.8435 - loss: 0.4716 - val_accuracy: 0.5000 - val_loss: 1.4895
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 82s 5s/step - accuracy: 0.7711 - loss: 0.5867 - val_accuracy: 0.5000 - val_loss: 1.6238
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
Training completed and model saved.
```

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_classes, target_names=SELECTED_CLASSES))

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=SELECTED_CLASSES, yticklabels=SELECTED_CLASSES)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Classification Report:
               precision    recall  f1-score   support

   Basketball       0.00      0.00      0.00         0
       Biking       0.67      0.67      0.67         3
PlayingGuitar       0.50      0.50      0.50         2
       Typing       0.50      0.50      0.50         2
     JumpRope       1.00      0.33      0.50         3

     accuracy                           0.50        10
    macro avg       0.53      0.40      0.43        10
 weighted avg       0.70      0.50      0.55        10

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



	Basketball	Biking	PlayingGuitar	Typing	JumpRope
Basketball	0	0	0	0	0
Biking	1	2	0	0	0
PlayingGuitar	1	0	1	0	0
Typing	0	0	1	1	0
JumpRope	0	1	0	1	1